**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Построение и анализ алгоритмов»**

# **Тема: Алгоритм Кнута-Морриса-Пратта.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8304 |  | Сергеев А.Д. |
| Преподаватель |  | Размочаева Н.В. |

Санкт-Петербург

2020

**Цель работы.**

Изучить алгоритм Кнута-Морриса-Пратта для оптимального поиска всех вхождений подстроки в строку.

**Задание.**

1. Реализуйте алгоритм КМП и с его помощью для заданных шаблона и текста найдите все вхождения в .

Вход:

Первая строка -

Вторая строка -

Выход:

Индексы начал вхождений в , разделенных запятой, если не входит в , то вывести -1.

1. Заданы две строки и .

Определить, является ли циклическим сдвигом (это значит, что и имеют одинаковую длину и состоит из суффикса , склеенного с префиксом ). Например, defabc является циклическим сдвигом abcdef.

Вход:

Первая строка -

Вторая строка -

Выход:

Если вляется циклическим сдвигом , индекс начала строки в , иначе вывести -1. Если возможно несколько сдвигов вывести первый индекс.

Дополнительное задание (вариант 1): Подготовка к распараллеливанию: работа по поиску разделяется на k равных частей, пригодных для обработки k потоками (при этом длина образца гораздо меньше длины строки поиска).

**Порядок выполнения работы.**

Написание работы производилось на базе операционной системы Windows 10 на языке программирования kotlin в среде программирования IntelliJ IDEA и на языке программирования с++ в среде программирования CLion.

В связи со спецификацией дополнительного задания (длина образца гораздо меньше длины строки поиска) выполнение подготовки к распараллеливанию возможно только для первой части работы, во второй в любом случае длина образца в два раза короче длины строки поиска.

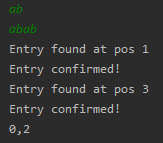
Первая программа состоит из трёх функций, расширяющих тип String. Функция prefix вычисляет значение префикс-функции для данной строки. Функция findAll реализует стандартный алгоритм Кнута-Морриса-Пратта для данной строки, строки образца и массива значений префикс-функции для данного образца. Она возвращает объект класса данных ListAndInt, содержащий в себе список вхождений образца в данную строку (переменная list) и целое число, равное длине максимального постфикса данной строки, являющегося одновременно префиксом образца (переменная int). Функция findAllSync делит данную строку на 100 частей, если образец короче её в 100 и более раз. Потом для каждой из полученных подстрок вызывается функция findAll, а их результаты объединяются. В случае, если одна или несколько объектов ListAndInt, полученных от findAll содержат переменную int больше 0, инициируется дополнительный поиск на подстроке, равной длине образца + int. Если образец короче данной строки менее чем в 100 раз, то вызывается функция findAll для всей данной строки целиком.

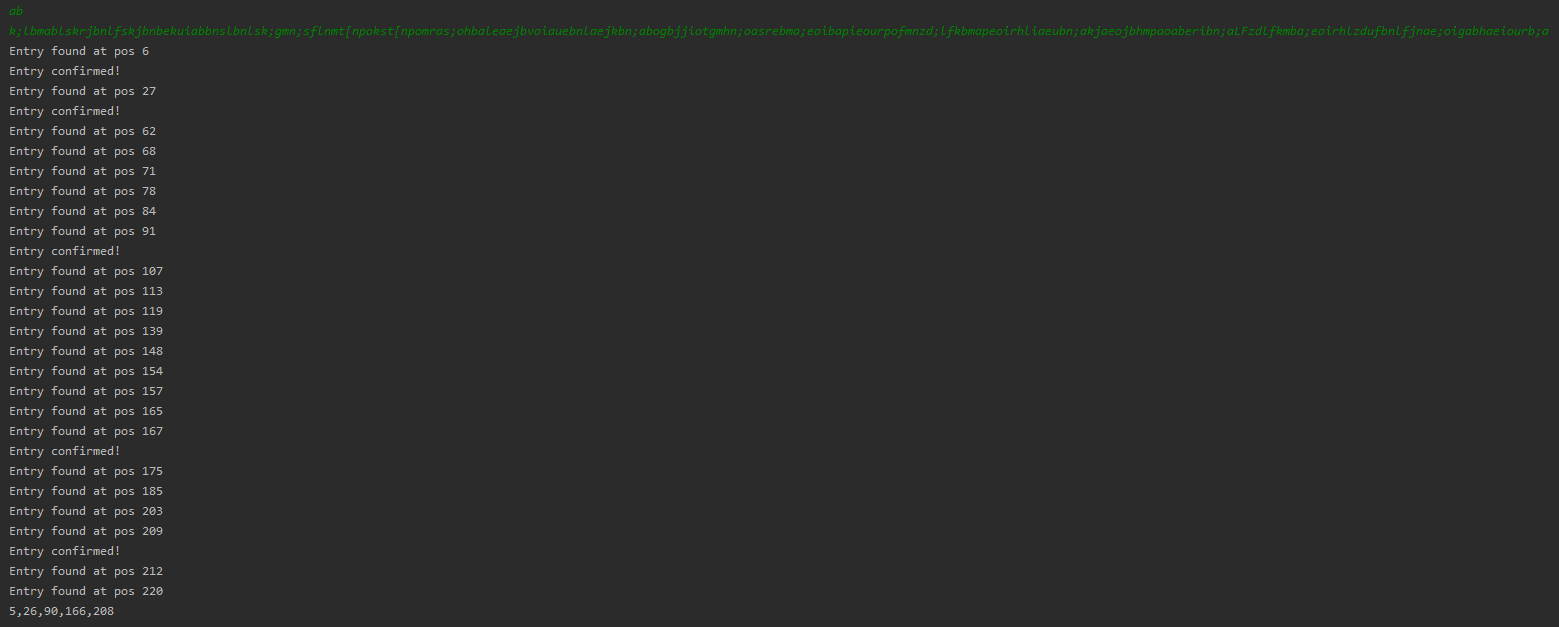
Вторая программа состоит из двух функций. Функция prefix вычисляет значение префикс-функции для данной строки. Функция isCyclic принимает на вход две строки и выполняет алгоритм Кнута-Морриса-Пратта, дважды проходя по строке поиска.

Описание классов в UML-виде приложено к отчеты в файлах UML.K.png и UML.C.png.

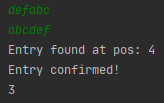
**Тестирование.**

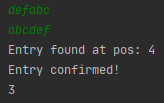
Тестирование первой программы:

Рисунок 1 - Вывод первой программы при образце короче строки поиска менее чем в 100 раз

Рисунок 2 -Вывод первой программы при образце короче строки поиска более чем в 100 раз

Тестирование второй программы:

Рисунок 3 - Вывод второй программы, если A является циклическим сдвигом B

Рисунок 4 - Вывод второй программы, если A не является циклическим сдвигом B

**Вывод.**

В результате лабораторной работы были изучены алгоритм Кнута-Морриса-Пратта и префикс-функция.

**Приложение А**

**Исходный код программы, файл Main.kt**

private data class ListAndInt(var list: List<Int>, var int: Int)

fun main() {

val mark = readLine()!!

val base = readLine()!!

val entryArray = base.findAllSync(mark)

println(if (entryArray.isEmpty()) "-1" else entryArray.joinToString(","))

}

private fun String.prefix() : IntArray {

val pi = IntArray(this.length)

var k = 0

for (q in 1 until this.length) {

if (k > 0 && this[k] != this[q]) k = pi[k]

if (this[k] == this[q]) k += 1

pi[q] = k

}

return pi

}

private fun String.findAll(mark : String, prf : IntArray, offset : Int) : ListAndInt {

val answer = mutableListOf<Int>()

var counter = 0

for (i in this.indices) {

while (counter > 0 && this[i] != mark[counter]) counter = prf[counter - 1]

if (this[i] == mark[counter]) {

if (counter == 0) println("Entry found at pos ${i + offset + 1}")

if (counter == mark.length - 1) println("Entry confirmed!")

counter++

}

if (counter == mark.length) {

answer.add(i - mark.length + 1)

counter = prf[counter - 1]

}

}

return ListAndInt(answer, counter)

}

fun String.findAllSync(mark : String) : IntArray {

val prefixes = mark.prefix()

return if (mark.length > this.length / 100) {

findAll(mark, prefixes, 0).list.toIntArray()

} else {

val ptL = mark.length \* 100

val answer = mutableListOf<Int>()

for (i in 0 until (this.length / ptL)) {

var subs = this.substring(i \* ptL, (i + 1) \* ptL)

var result = subs.findAll(mark, prefixes, i \* ptL)

answer.addAll(result.list.map { it + i \* ptL })

val offset = result.int

val tailLen = offset + mark.length - 1

subs = this.substring((i + 1) \* ptL - offset, ((i + 1) \* ptL - offset + tailLen).coerceAtMost(this.length))

result = subs.findAll(mark, prefixes, (i + 1) \* ptL - offset)

answer.addAll(result.list.map { it + i \* ptL + ptL - offset })

}

val result = this.substring(this.length - (this.length % ptL)).findAll(mark, prefixes, this.length - (this.length % ptL))

answer.addAll(result.list.map { it + this.length - (this.length % ptL) })

answer.toIntArray()

}

}

**Приложение Б**

**Исходный код программы, файл main.cpp**

#include <iostream>

void prefix(std::string& mark, int\*\* prefix) {

int k = 0;

(\*prefix)[0] = 0;

for (int i = 0; i < mark.size(); ++i) {

if (k > 0 && mark[k] != mark[i]) k = (\*prefix)[k];

if (mark[k] == mark[i]) k += 1;

(\*prefix)[i] = k;

}

}

int isCyclic(std::string& a, std::string& b) {

int\* prf = ((int\*) (calloc(b.size(), sizeof(unsigned int\*))));

prefix(b, &prf);

int counter = 0;

for (int i = 0; i < a.size() \* 2; ++i) {

if (counter > 0 && a[i % a.size()] != b[counter]) counter = prf[counter - 1];

if (a[i % a.size()] == b[counter]) {

if (!counter) std::cout << "Entry found at pos: " << i % (a.size()) + 1 << std::endl;

if (counter == a.size() - 1) std::cout << "Entry confirmed!" << std::endl;

counter++;

}

if (counter == b.size()) {

return i - a.size() + 1;

}

}

return -1;

}

int main() {

std::string a = std::string(), b = std::string();

std::cin >> a;

std::cin >> b;

if (a.size() != b.size()) {

std::cout << -1;

} else {

std::cout << isCyclic(a, b);

}

}